# 1- Image Encryption Code :

It consists of 3 functions:

1. `public static void Get_Password(ref int Seed, int Tape, int n)`
2. `public static string AlphaNumericPW (string s)`
3. `public static void Encrypt_Decrypt(ref RGBPixel[,] ImageMatrix, string seed, int tape)`

1. Get_Password function:

It takes 3 parameters , The First parameter is the (Seed) which is the initial seed and it is sent by reference to change its value in all the functions related to it, we use it to get a Key after shifting it using the Tape Position Given in the Second parameter (Tape) and XOR it with Red Component, Green Component and Blue Component of the input image and Third Parameter is (n) Which is (Seed Length - 1) , it is of O(1) .

```
public static void Get_Password(ref Int64 Seed, int Tape, int n) //θ(1)
        {
            Int64 newBit, TAPEv, MSBv, X = 1; //o(1)-Assignment
            for (int z = 0; z < 8; z++) //o(1)
            {
                MSBv = Seed; //o(1) -assignment
                MSBv = MSBv & (X << n); //o(1)-(assignment , shift left ,and)
                MSBv = MSBv >> n; //o(1)-shift right
                TAPEv = Seed; //o(1) -assignment
                TAPEv = TAPEv & (X << Tape); //o(1)-(assignment , shift left,and)
                TAPEv = TAPEv >> Tape; //o(1)-(assignment , shift right)
                newBit = MSBv ^ TAPEv; //o(1)-(assignment , xor)
                Seed = Seed << 1; //o(1)-(assignment , shift left)
                Seed = Seed | newBit; //o(1)-(assignment , or)
            }
        }        }
```

➔ This Function is Used to Implement (LFSR) Algorithm, by getting the MSB value and the Tape Position Value and XOR them to generate newBit after that we shift the Seed and or it with the newBit.

2. AlphaNumericPW function:

It takes 1 parameter which is the initial seed Entered by the user to convert any Alphabetic Character into binary value and check that all values is 0's and 1's and return a Binary value as an initial seed.

```
public static string AlphaNumericPW(string s)//θ(s.Length)
        {
            int chk = 0;    //o(1)-Assignment
            for (int i = 0; i < s.Length; i++)  //o(s.Length)
            {
                if (s.ElementAt(i).Equals('0') || s.ElementAt(i).Equals('1'))
                    chk++;  //o(1)
            }
            if (chk != s.Length)
            {
                string result = string.Empty;   //o(1)-Assignment
                foreach (char ch in s)  //o(s.Length)
                {
                    result += Convert.ToString((int)ch, 2); //o(1)
                }
                if (result.Length > 63)
                    result = result.Substring(result.Length - 63, 63);
                return result;
            }
            return s;
        }
```

3. Encrypt_Decrypt function:

It takes 3 parameters from the user in the GUI (Image, initial seed, tape position) first parameter is the image matrix of the image the second is initial seed and third parameter is the tape position it calls 2 functions, AlphaNumericPW to Check the initial seed if it is 0's and 1's or not or Convert the initial seed from alphabets to binary numbers and then it calls the function Get_Password to Encrypt or Decrypt Every Pixel in the image , it is of O(h*w).

```
public static void Encrypt_Decrypt(ref RGBPixel[,] ImageMatrix, string seed, int tape)  // o(h*w) +
o(s.length)
        {
            seed = AlphaNumericPW(seed);    //o(s.length)
            Int64 se = Convert.ToInt32(seed, 2); //o(1) -(assignment , convert)
            int sLen = seed.Length - 1; //o(1) -Assignment
            for (int i = 0; i < GetHeight(ImageMatrix); i++) //o(h*w)
            {
                for (int j = 0; j < GetWidth(ImageMatrix); j++) //o(w)
                {
                    //=========Encrypt/Decrypt Red Component=========

                    Get_Password(ref se, tape, sLen); //o(1)
                    Int64 w = se & 255; // o(1) -(Assignment && and operator)
                    ImageMatrix[i, j].red = (byte)((ImageMatrix[i, j].red) ^ w); //o(1) (Assignment
&& xor)
                    //=========Encrypt/Decrypt Green Component=========
                    Get_Password(ref se, tape, sLen); //o(1)
                    w = se & 255;  // o(1) -(Assignment && and operator)
                    ImageMatrix[i, j].green = (byte)((ImageMatrix[i, j].green) ^ w); //o(1)
(Assignment && xor)
                    //   =========Encrypt/Decrypt Blue Component=========
                    Get_Password(ref se, tape, sLen); //o(1)
                    w = se & 255;  // o(1) -(Assignment && and operator)
                    ImageMatrix[i, j].blue = (byte)((ImageMatrix[i, j].blue) ^ w); //o(1) (Assignment
&& xor)
                }
            }
        }
```

## 2- Constructing the Huffman Tree Code:

It consists of 8 functions:

```
1.  public static void Min_Heapify(ref int h, List<Node> a, int i)
2.  public static void Build_MinHeap(List<Node> a)
3.  public static Node Extract_HeapMin(List<Node> a, ref int h)
4.  public static void Heap_Decrease_key(List<Node> a, int i, Node key)
5.  public static void Min_Heap_Insert(List<Node> a, Node key, ref int h)
6.  public static Node Build_Huffman(List<Node> Component)
7.  public static void Save_Tree(Node Root, StreamWriter sw, string BinaryVal)
8.  public static void Huffman_Code(ref RGBPixel[,] ImageMatrix)
```

1. Min_Heapify:
   It takes 3 parameters the first is ( h ) which is the heap length , the second is a list of nodes ( a ) and the third is ( i ) refers to the indices it is a recursive function , it if of O(Log n) and its recuersive takes `//T(N) <= T(2 N / 3) + θ (1)`.

```
        public static void Min_Heapify(ref int h, List<Node> a, int i) // o(log N)
        {
         int large_value = -1; // o(1) assignment
         int left = (2 * (i + 1)) - 1; // o(1) assignment
         int right = (2 * (i + 1)); // o(1) assignment
         if ((left < h) && (a[left].Frequency < a[i].Frequency)) // o(1)
          {
            large_value = left; // o(1) assignment
          }
         else
           {
            large_value = i; // o(1) assignment
           }
         if ((right < h) && (a[right].Frequency < a[i].Frequency)) // o(1)
          {
            large_value = right; // o(1) assignment
          }
         if (large_value != i) // o(1) assignment
          {
           Node temp = a[i]; // o(1) (assignment && retrive index in array)
           a[i] = a[large_value]; //o(1)(assignment && retrive index in array)
           a[large_value] = temp; //o(1)(assignment && retrive index in array)
           Min_Heapify(ref h, a, large_value); //T(2 N / 3)
             //T(N) <= T(2 N / 3) + θ (1)
          }

        }       }
```

2. Build_MinHeap:
   It takes 1 parameter which is (a) List of nodes and sorts them in the form of heap and it calls Min_Heapify function to get the Next Nodes , it is of O(n Log n).

```
public static void Build_MinHeap(List<Node> a) // o(Nlog N)
{
   int heap_size = a.Count(); //o(1)-Assignment
   for (int i = ((a.Count()) / 2); i >= 0; i--) //o(N)
  {
    Min_Heapify(ref heap_size, a, i); //O(log N)
  }
}
```

3. Extract_HeapMin:

   It takes 2 parameters a List and Index; it is of O (Log n).

```
public static Node Extract_HeapMin(List<Node> a, ref int h)//o(log N)
     {
         Node Min = a[0]; //o(1)-(Assignment && reterive value from array
         a[0] = a[h - 1]; //o(1)-(Assignment && reterive value from array
         h = h - 1; //o(1)-(Assignment)
         Min_Heapify(ref h, a, 0); //o(log N)
         return Min; //o(1)
     }
```

4. Heap_Decrease_key:
   It is of O (Log n).

```
public static void Heap_Decrease_key(List<Node> a, int i, Node key) //o(log N)
 {
   a[i] = key; //o(1)-Assignment
  while ((i > 0) && (a[i / 2].Frequency > a[i].Frequency)) //o(Log N) AS IN EACH
TIME WE DIVIE BY 2
  {
    Node temp = a[i]; //o(1)-(Assignment && reterive value from array
    a[i] = a[i / 2]; //o(1)-(Assignment && reterive value from array
    a[i / 2] = temp; //o(1)-(Assignment && reterive value from array
    i = i / 2;//o(1)-Assignment
         }
     }
```

5. Min_Heap_Insert:
   It is of O (Log n).

```
public static void Min_Heap_Insert(List<Node> a, Node key, ref int h)
//o(log N)
        {
      h = h + 1; //o(1)-Assignment
     Heap_Decrease_key(a, h - 1, key); //o(log N)
        }
    }
```

6. Build_Huffman:
   It is O (nLog n)

```
public static Node Build_Huffman(List<Node> Component) //o(Nlog N)
       {
  Priority_Queue.Build_MinHeap(Component);//o(log N)
  int C_Heap = Component.Count();//o(1)-Assignment
           int n = Component.Count() - 1;//o(1)-Assignment
           for (int i = 0; i < n; i++) //o(N) *o(logN)
           {
               Node temp = new Node(0, 0); //o(1)
               Node right = Priority_Queue.Extract_HeapMin(Component, ref       C_Heap);//o(log N)
               Node left = Priority_Queue.Extract_HeapMin(Component, ref C_Heap);//o(log N)
               temp.Frequency = (left.Frequency + right.Frequency); //o(1)-Assignment
               temp.Left = left; //o(1)-Assignment
               temp.Right = right; //o(1)-Assignment
               Priority_Queue.Min_Heap_Insert(Component, temp, ref C_Heap);//o(log N)
           }
           return Priority_Queue.Extract_HeapMin(Component, ref C_Heap); //o(log N)
       }
```

7. Save_Tree:

It is of O(log n) AND the recuersive relation is as T(N)=N/2+θ

```
public static void Save_Tree(Node Root, StreamWriter sw, string BinaryVal, string[] arr, ref long
length) //o(log N) as the recuersive relation is as T(N)=N/2+θ
        {
            if (Root.Right != null) Save_Tree(Root.Right, sw, BinaryVal + '1', arr, ref length);

            if (Root.Left != null) Save_Tree(Root.Left, sw, BinaryVal + '0', arr, ref length);

            if ((Root.Left == null) && (Root.Right == null))//o(1)
            {

                sw.WriteLine(Convert.ToString(Root.Value) + " " + BinaryVal + " " +
Convert.ToString(Root.Frequency) + " " + Convert.ToString((BinaryVal.Length) * (Root.Frequency)));
//o(1)
                Total_Bits += ((BinaryVal.Length) * (Root.Frequency)); //o(1)-(addition&&
assignment)
                length += ((BinaryVal.Length) * (Root.Frequency)); //o(1)-(addition&& assignment)
            }
```

8. Huffman_Code:

It is of O(h*w).

```
441         public static void Huffman_Code(ref RGBPixel[,] ImageMatrix)
442         {
443             matrix_dimintion = ((GetHeight(ImageMatrix)) * (GetWidth(ImageMatrix))) * 24;
444             long rem = 0; //o(1)-Assignment
445             List<Node> RedL = new List<Node>();
446             List<Node> GreenL = new List<Node>();
447             List<Node> BlueL = new List<Node>();
448             Total_Bits = red_bytes = green_bytes = blue_bytes = 0;//o(1)-Assignment
449             R = new int[256]; //o(1)-(Assignment - new)
450             G = new int[256]; //o(1)-Assignment-new)
451             B = new int[256]; //o(1)-Assignment-new
452             for (int i = 0; i < GetHeight(ImageMatrix); i++) //o(h*w)
453             {
454                 for (int j = 0; j < GetWidth(ImageMatrix); j++) //o(w)
455                 {
456                     R[Convert.ToInt32(ImageMatrix[i, j].red)] += 1; //o(1)-(Assignment-convert-addition-
457                     G[Convert.ToInt32(ImageMatrix[i, j].green)] += 1; //o(1)-(Assignment-convert-addition-
458                     B[Convert.ToInt32(ImageMatrix[i, j].blue)] += 1; //o(1)-(Assignment-convert-addition-
459                 }
460             }
461             for (int i = 0; i < 256; i++)//o(1)
462             {
463                 if (R[i] != 0)
464                 { Node x = new Node(i, R[i]); RedL.Add(x); }    //o(1)-(Assignment - new) & (Add the end of list)
465                 if (G[i] != 0)
466                 { Node x = new Node(i, G[i]); GreenL.Add(x); }  //o(1)-(Assignment - new) & (Add the end of list)
467                 if (B[i] != 0)
468                 { Node x = new Node(i, B[i]); BlueL.Add(x); }   //o(1)-(Assignment - new) & (Add the end of list)
469 
470             }
471             red_length = RedL.Count(); //o(1)-Assignment
472             green_length = GreenL.Count(); //o(1)-Assignment
473             blue_length = BlueL.Count(); //o(1)-Assignment
474             arr_red = new string[256]; //o(1)-Assignment && new
475             arr_gr = new string[256];//o(1)-Assignment && new
476             arr_bl = new string[256];//o(1)-Assignment && new
477             FileStream fs = new FileStream("RGB-Tree.txt", FileMode.Truncate);
478             StreamWriter sw = new StreamWriter(fs);
479             string BinVal = "";
480             Node RTRoot = Build_Huffman(RedL); //o(Nlog N)
481             Save_Tree(RTRoot, sw, BinVal, arr_red, ref red_bytes); //o(log n)
482             rem = red_bytes % 8; //o(1)-assignment
483             red_bytes = red_bytes / 8; //o(1)-assignment
484             if (rem != 0)
485             {
486                 red_bytes += 1; //o(1)-assignment
487             }
488 
489             // temp.Clear();
490             sw.WriteLine("============================");
491             Node GTRoot = Build_Huffman(GreenL);//o(Nlog N)
492             Save_Tree(GTRoot, sw, BinVal, arr_gr, ref green_bytes);//o(log n)
493             rem = green_bytes % 8;  //o(1)-assignment
494             green_bytes = green_bytes / 8;  //o(1)-assignment
495             if (rem != 0)
496             {
497                 green_bytes += 1;   //o(1)-assignment
498             }
499 
500             // temp.Clear();
501             sw.WriteLine("============================");
502             Node BTRoot = Build_Huffman(BlueL);//o(Nlog N)
503             Save_Tree(BTRoot, sw, BinVal, arr_bl, ref blue_bytes);//o(log n)
504             rem = blue_bytes % 8;   //o(1)-assignment
505             blue_bytes = blue_bytes / 8;    //o(1)-assignment
506             if (rem != 0) //o(1)-assignment
507             {
508                 blue_bytes += 1;    //o(1)-assignment
509             }
510 
511             // temp.Clear();
512             long res = (Total_Bits) / 8;    //o(1)-assignment
513             sw.WriteLine(Convert.ToString(res));//o(1)(convert && writ in file)
514             float ratio = (Total_Bits) / (ImageOperations.matrix_dimintion) * 100;  //o(1)-assignment
515             sw.WriteLine(Convert.ToString(ratio) + " % ");//o(1)(convert && writ in file)
516             sw.Close();
517             fs.Close();
518         }
```

# 3-compreesion

It consists of 2 functions:

17. `public static void str_1(RGBPixel[,] ImageMatrix)`
18. `public static void compress_image(int[] red_freq, int[] green_freq, int[] blue_freq, byte[] red_com, byte[] green_com, byte[] blue_com, int tape, string seed, int w, int h).`

## 1- Str_1

```
304     public static void str_1(RGBPixel[,] ImageMatrix)//O(h*w)
305     {
306         arr = new byte[red_bytes]; //O(1) (assignment && new].
307         arr1 = new byte[green_bytes]; //O(1) (assignment && new].
308         arr2 = new byte[blue_bytes]; //O(1) (assignment && new].
309         int ar = 8, ag = 8, ab = 8; //O(1)-(assignment)
310         int index_red = 0, index_green = 0, index_blue = 0; //o(1)-(assignment)
311         string temp, temp1, temp2, rf, rf1, rf2;
312         for (int i = 0; i < GetHeight(ImageMatrix); i++) //o(h*w)
313         {
314             for (int j = 0; j < GetWidth(ImageMatrix); j++) //o(w)
315             {
316                 temp = arr_red[ImageMatrix[i, j].red]; //o(1)-assignment
317                 if (temp.Length < ar)//o(1)
318                 {
319                     arr[index_red] <<= temp.Length; //o(1)-(put index in array && shift)
320                     arr[index_red] += Convert.ToByte(temp, 2); //o(1)-(put index in array && assignment && addition && convert)
321                     ar -= temp.Length; //o(1)-subtraction && assignment
322                 }
323                 else if (temp.Length == ar)
324                 {
325                     arr[index_red] <<= temp.Length; ////o(1)-(put index in array && shift)
326                     arr[index_red] += Convert.ToByte(temp, 2); //o(1)-(put index in array && assignment && addition && convert)
327                     index_red++; // o(1)-addition && assignment
328                     ar = 8; //o(1)-assignment
329                 }
330                 else
331                 {
332                     rf = temp.Substring(0, ar);  //o(1) - assignment && substring
333                     arr[index_red] <<= ar; //o(1)-(put index in array && shift)
334                     arr[index_red] += Convert.ToByte(rf, 2);  //o(1)-(put index in array && assignment && addition && convert)
335                     index_red++; //o(1)-addition && assignment
336                     temp = temp.Substring(ar, temp.Length - ar);//o(1) - assignment && substring
337 
338                     while (temp.Length >= 8) //o(1) AS temp size is limited to 32
339                     {
340                         rf = temp.Substring(0, 8); //o(1)-assignment && substring
341                         arr[index_red] <<= 8; //o(1)-(put index in array && shift)
342                         arr[index_red] += Convert.ToByte(rf, 2); //o(1)-(put index in array && assignment && addition && convert)
343                         index_red++; //o(1)-addition && assignment
344                         temp = temp.Substring(8, temp.Length - 8); //o(1) - assignment && substring
345                     }
346                     if (temp.Length != 0) //o(1)
347                     {
348                         arr[index_red] <<= temp.Length; //o(1)-(put index in array && shift)
349                         arr[index_red] += Convert.ToByte(temp, 2); //o(1)-(put index in array && assignment && addition && convert)
350                         ar = 8 - temp.Length; // o(1) - assignment
351                     }
352                     else
353                         ar = 8; //o(1) - assignment
354                 }
355                 temp1 = arr_gr[ImageMatrix[i, j].green]; //o(1)-assignment
356                 if (temp1.Length < ag) //o(1)
357                 {
358                     arr1[index_green] <<= temp1.Length; //o(1)-(put index in array && shift)
359                     arr1[index_green] += Convert.ToByte(temp1, 2);//o(1)-(put index in array && assignment && addition && convert)
360                     ag -= temp1.Length; //o(1)-subtraction && assignment
361                 }
362                 else if (temp1.Length == ag)
363                 {
364                     arr1[index_green] <<= temp1.Length; //o(1)-(put index in array && shift)
365                     arr1[index_green] += Convert.ToByte(temp1, 2); //o(1)-(put index in array && assignment && addition && convert)
366                     index_green++; //o(1)-addition && assignment
367                     ag = 8; //o(1)-assignment
368                 }
```

```
                else
                {
                    rf1 = temp1.Substring(0, ag);//o(1) - assignment && substring
                    arr1[index_green] <<= ag; //o(1)-(put index in array && shift)
                    arr1[index_green] += Convert.ToByte(rf1, 2); //o(1)-(put index in array && assignment && addition && convert)
                    index_green++; // o(1)-addition && assignment
                    temp1 = temp1.Substring(ag, temp1.Length - ag);//o(1) - assignment && substring
                    while (temp1.Length >= 8) //o(1) AS temp size is limited to 32
                    {
                        rf1 = temp1.Substring(0, 8);//o(1)-assignment && substring
                        arr1[index_green] <<= 8; //o(1)-(put index in array && shift)
                        arr1[index_green] += Convert.ToByte(rf1, 2);//o(1)-(put index in array && assignment && addition && convert)
                        index_green++; //o(1)-addition && assignment
                        temp1 = temp1.Substring(8, temp1.Length - 8); //o(1) - assignment && substring
                    }
                    if (temp1.Length != 0) //o(1)
                    {
                        arr1[index_green] <<= temp1.Length; //o(1)-(put index in array && shift)
                        arr1[index_green] += Convert.ToByte(temp1, 2); //o(1)-(put index in array && assignment && addition && convert)
                        ag = 8 - temp1.Length; // o(1) - assignment
                    }
                    else
                        ag = 8; // o(1) - assignment
                }
                temp2 = arr_bl[ImageMatrix[i, j].blue]; //o(1)-assignment
                if (temp2.Length < ab) //o(1)
                {
                    arr2[index_blue] <<= temp2.Length; //o(1)-(put index in array && shift)
                    arr2[index_blue] += Convert.ToByte(temp2, 2);//o(1)-(put index in array && assignment && addition && convert)
                    ab -= temp2.Length; //o(1)-subtraction && assignment
                }
                else if (temp2.Length == ab)
                {
                    arr2[index_blue] <<= temp2.Length; //o(1)-(put index in array && shift)
                    arr2[index_blue] += Convert.ToByte(temp2, 2); //o(1)-(put index in array && assignment && addition && convert)
                    index_blue++;//o(1)-addition && assignment
                    ab = 8; //o(1)-assignment
                }
                else
                {
                    rf2 = temp2.Substring(0, ab);//o(1) - assignment && substring
                    arr2[index_blue] <<= ab;//o(1)-(put index in array && shift)
                    arr2[index_blue] += Convert.ToByte(rf2, 2); //o(1)-(put index in array && assignment && addition && convert)
                    index_blue++; //o(1)-addition && assignment
                    temp2 = temp2.Substring(ab, temp2.Length - ab); //o(1) - assignment && substring

                    while (temp2.Length >= 8) //o(1) AS temp size is limited to 32
                    {
                        rf2 = temp2.Substring(0, 8); //o(1)-assignment && substring
                        arr2[index_blue] <<= 8;//o(1)-(put index in array && shift)
                        arr2[index_blue] += Convert.ToByte(rf2, 2);//o(1)-(put index in array && assignment && addition && convert)
                        index_blue++;//o(1)-addition && assignment
                        temp2 = temp2.Substring(8, temp2.Length - 8);//o(1) - assignment && substring
                    }
                    if (temp2.Length != 0) //o(1)
                    {
                        arr2[index_blue] <<= temp2.Length; //o(1)-(put index in array && shift)
                        arr2[index_blue] += Convert.ToByte(temp2, 2); //o(1)-(put index in array && assignment && addition && convert)
                        ab = 8 - temp2.Length; // o(1) - assignment
                    }
                    else
                        ab = 8; // o(1) - assignment
                }
            }

        }
```

## 2- Compress image

```
        public static void compress_image(int[] red_freq, int[] green_freq, int[] blue_freq, byte[] red_com,
byte[] green_com, byte[] blue_com, int tape, string seed, int w, int h)//o(Nlog N)
        {
            byte[] redd = new byte[1024];//o(1) (assignment)
            byte[] grenn = new byte[1024];//o(1) (assignment)
            byte[] bluee = new byte[1024];//o(1) (assignment)
            for (int i = 0; i < 256; i++)
            {
                Array.Copy(BitConverter.GetBytes(red_freq[i]), 0, redd, i * 4, 4);//o(nlon)(number of iterations*4
&& copy to array)
                Array.Copy(BitConverter.GetBytes(green_freq[i]), 0, grenn, i * 4, 4);//o(nlon)(number of
iterations*4 && copy to array)
                Array.Copy(BitConverter.GetBytes(blue_freq[i]), 0, bluee, i * 4, 4);//o(nlon)(number of
iterations*4 && copy to array)
            }
            FileStream ffs = new FileStream("com.txt", FileMode.Truncate);
            StreamWriter ffss = new StreamWriter(ffs);
            ffss.WriteLine(red_com.Length);//o(1) (write in file)
            ffss.WriteLine(green_com.Length);//o(1) (write in file)
            ffss.WriteLine(blue_com.Length);//o(1) (write in file)
            ffss.Close();
            ffs.Close();

            FileStream ss = new FileStream("compressed.txt", FileMode.Truncate);
            BinaryWriter bwr = new BinaryWriter(ss);
            bwr.Write(redd);//o(1) (write in file)
            bwr.Write(grenn); //o(1) (write in file)
            bwr.Write(bluee);//o(1) (write in file)
            bwr.Write(red_com);//o(1) (write in file)
            bwr.Write(green_com);//o(1) (write in file)
            bwr.Write(blue_com);//o(1) (write in file)
            bwr.Write(seed);//o(1) (write in file)
            bwr.Write(tape);//o(1) (write in file)
            bwr.Write(w);//o(1) (write in file)
            bwr.Write(h);//o(1) (write in file)
            bwr.Close();
            ss.Close();
        }
```

## 4-decompreesion

It consists of 3 functions:

19. `public static Node get_next_node(byte p, Node n)`
20. `public static List<int> decompress(byte[] c, Node root)`
21. `public static RGBPixel[,] decompress_image()`

# 1- get next node

```
public static Node get_next_node(byte p, Node n)//o(1)return the Next node to get path
{
    if (p == 0)//o(1) Assigment
    { return n.Left; }//o(1)return
    else
        return n.Right;//o(1)return
}
```

# 2- decmopress

```
public static List<int> decompress(byte[] c, Node root)//o(n) get the list of specfiec color
{
    List<int> temp = new List<int>();//o(1) define a new list that save the color values
    byte var = 128;//o(1) assigment to get the bits from the byte color
    int accu = 0;// o(1) assigment that count to 8 to get the value of one byte
    Node f = root;// o(1) assigment that point to the top node in huffman
    for (int i = 0; i < c.Length;)//o(s,length)
    {
        while (accu < 8)//o(1) this loop count untill the byte read all
        {
            byte aa = (byte)(c[i] & var);//o(1) assigmrnt to get the spceific bit
            Node check = get_next_node(aa, f);//o(1) call the function to get the next node according to
the bit value
            if (check.Left == null && check.Right == null)//o(1) check that the cuurent node is a leaf
node
            {
                temp.Add(check.Value);// o(1) add to the list the value of the cuurent color
                f = root;// O(1) make search start from the root or the huffman
            }
            else

                f = check;//o(1)make the start node is the current node

            var /= 2;// o(1)  divide the var to get the next bit
            accu++;// o(1) read another 8 bits

        }
        i++;// o(1) go to the next list value to save on it
        accu = 0;// o(1) reset the counter
        var = 128;// o(1) reset the var to make it point to the first bit
    }
    return temp;//o(1) return the list that contain the colors values
}
```

# 2- decmopress image

```
public static RGBPixel[,] decompress_image()//o(h*w)
{
    FileStream fo = new FileStream("com.txt", FileMode.Open);
    StreamReader ffo = new StreamReader(fo);
    int red_l = Convert.ToInt32(ffo.ReadLine());//o(1)read the red_bytes length
    int green_l = Convert.ToInt32(ffo.ReadLine());//o(1) read the green bytes length
    int blue_l = Convert.ToInt32(ffo.ReadLine());//o(1) read the blue bytes length
    ffo.Close();
```

```
            fo.Close();
            FileStream fs = new FileStream("compressed.txt", FileMode.Open);
            BinaryReader br = new BinaryReader(fs);
            byte[] red_freq = br.ReadBytes(1024);
            byte[] green_freq = br.ReadBytes(1024);
            byte[] blue_freq = br.ReadBytes(1024);
            int[] red1 = new int[256];
            int[] green1 = new int[256];
            int[] blue1 = new int[256];
            List<Node> red_frq = new List<Node>();
            List<Node> green_frq = new List<Node>();
            List<Node> blue_frq = new List<Node>();

            for (int i = 0; i < 1024; i += 4)//o(1) this loop take 4 bytes and compress them to
one int32 value
            {
                red1[i / 4] = BitConverter.ToInt32(red_freq, i);
                green1[i / 4] = BitConverter.ToInt32(green_freq, i);
                blue1[i / 4] = BitConverter.ToInt32(blue_freq, i);
            }
            for (int j = 0; j < 256; j++)//o(1) this loop add in the lists that it's value is
not 0
            {
                if (red1[j] != 0)
                { Node x = new Node(j, red1[j]); red_frq.Add(x); }
                if (green1[j] != 0)
                { Node x = new Node(j, green1[j]); green_frq.Add(x); }
                if (blue1[j] != 0)
                { Node x = new Node(j, blue1[j]); blue_frq.Add(x); }

            }
            Node r1 = Build_Huffman(red_frq);//o(n logn)
            Node r2 = Build_Huffman(green_frq);//o(n logn)
            Node r3 = Build_Huffman(blue_frq);//o(n logn)
            byte[] red_co = br.ReadBytes(red_l);//o(1) read from the file the red bytes
compressed values
            byte[] green_co = br.ReadBytes(green_l);//o(1) read from the file the green bytes
compressed values
            byte[] blue_co = br.ReadBytes(blue_l);//o(1) read from the file the blue bytes
compressed values
            string seed = br.ReadString();// o(1) get the seed from the file
            int TAPE = br.ReadInt32();//o(1 )get the tape from the file
            int width = br.ReadInt32();// o(1) get the width from the file
            int heigth = br.ReadInt32();// o(1) get the heigth from the file
            br.Close();
            fs.Close();
            Tape_Position = TAPE;//o(1) save it to he global value
            Initial_Seed = seed;//o(1) save it to the global value
            List<int> redd = decompress(red_co, r1);//o(n) call the function that get the
acctual values of red
            List<int> grenn = decompress(green_co, r2);//o(n)call the function that get the
acctual values of green
            List<int> bluee = decompress(blue_co, r3);// o(n)call the function that get the
acctual values of blue
            RGBPixel[,] com_image = new RGBPixel[heigth, width];// create a new RGBpixel that
get the image values
            int index = 0;
            // this nessted loop create the iamge that displayed to the user
```

```
    for (int i = 0; i < heigth; i++)//o(H)
    {
        for (int j = 0; j < width; j++)//o(W)
        {
            com_image[i, j].red = (byte)redd[index];
            com_image[i, j].green = (byte)grenn[index];
            com_image[i, j].blue = (byte)bluee[index];
            index++;
        }
    }

    return com_image;//o(1) retun the image to the user
}
```